UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO DE CIÊNCIAS, TECNOLOGIAS E SAÚDE DO CAMPUS ARARANGUÁ
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

Davi Feliciano Nonnenmacher

Desenvolvimento de um protótipo de um ambiente integrado baseado em um robô educacional visando o ensino de conceitos básicos de programação

Araranguá
2022

Davi Feliciano Nonnenmacher

Desenvolvimento de um protótipo de um ambiente integrado baseado em um robô educacional visando o ensino de conceitos básicos de programação

Trabalho de Conclusão de Curso do Curso de Graduação em Engenharia de Computação do Centro de Ciências, Tecnologias e Saúde do Campus Araranguá da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Engenharia de Computação.
Orientador: Prof. Jim Lau, Dr.

Araranguá
2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Nonnenmacher, Davi Feliciano
Desenvolvimento de um protótipo de um ambiente integrado baseado em um robô educacional visando o ensino de conceitos básicos de programação / Davi Feliciano Nonnenmacher ; orientador, Jim Lau , 2022.
47 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Araranguá, Graduação em Engenharia de Computação, Araranguá, 2022.

Inclui referências.

1. Engenharia de Computação. 2. Programação em Blocos. 3. Robótica Educacional. 4. Gamificação. I. , Jim Lau. II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Computação. III. Título.

Davi Feliciano Nonnenmacher

Desenvolvimento de um protótipo de um ambiente integrado baseado em um robô educacional visando o ensino de conceitos básicos de programação

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Engenharia de Computação" e aprovado em sua forma final pelo Curso de Graduação em Engenharia de Computação.

Araranguá, 22 de março de 2022.

Documento assinado digitalmente
Analucia Schiaffino Morales
Data: 22/03/2022 18:42:42-0300
CPF: 622.256.420-87
Verifique as assinaturas em https://v.ufsc.br

Analúcia Schiaffino Morales, Dra.
Coordenadora do Curso

Banca Examinadora:

Documento assinado digitalmente
Jim Lau
Data: 22/03/2022 17:00:36-0300
CPF: 613.464.702-00
Verifique as assinaturas em https://v.ufsc.br

Prof. Jim Lau, Dr.
Orientador

Documento assinado digitalmente
ANALUCIA SCHIAFFINO MORALES
Data: 22/03/2022 18:43:26-0300
CPF: 622.256.420-87
Verifique as assinaturas em https://v.ufsc.br

Profa. Analúcia Schiaffino Morales, Dra.
Avaliadora
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Luciana Bolan Frigo
Data: 22/03/2022 17:42:42-0300
CPF: 910.389.799-00
Verifique as assinaturas em https://v.ufsc.br

Profa. Luciana Bolan Frigo, Dra.
Avaliadora
Universidade Federal de Santa Catarina

AGRADECIMENTOS

Dedico esse trabalho a meus pais, cujo apoio foi imprescindível ao longo da minha graduação. Sou grato também a todas as pessoas com quem estive em contato no campus UFSC - Araranguá, em especial aos professores e colegas de curso.

RESUMO

Este trabalho descreve um protótipo de um ambiente integrado para o robô de um projeto brasileiro de robótica educacional, bem como discute seu potencial como ferramenta para o ensino de conceitos básicos de programação. O objetivo deste trabalho é obter um ferramental de baixo custo que integre programação em blocos com robótica educacional com o intuito de que seja usado para o processo de ensino-aprendizagem de habilidades de programação e pensamento lógico em crianças e jovens, ou, de forma mais genérica, iniciantes em programação. O hardware em que o ambiente se baseia é de baixo custo em comparação às alternativas no mercado, e mesmo em sua ausência uma versão virtual do robô poderá ser utilizada, tendo nela sido incorporados aspectos de gamificação voltados ao ensino de conceitos básicos de programação.O robô utilizado baseia-se no robô do projeto Jabuti Edu, da UFRGS, e foi tomado como ponto de princípio para o desenvolvimento deste trabalho. O desenvolvimento foi guiado por, e apoia-se em, pesquisas na literatura científica envolvendo robótica educacional, programação em blocos e gamificação. Através dos esforços relatados no desenvolvimento, foi possível atingir os objetivos e disponibilizar o ambiente em formato *web*.

Palavras-chave: Programação em blocos, robótica educacional, gamificação.

ABSTRACT

This work describes a prototype for an integrated environment for a brazilian educational robotics project, as well as discusses its potential as a tool for teaching basic programming concepts. The objective is to provide low-cost tooling that integrates block programming with educational robotics with the intent of it being used for the process of teaching logical programming abilities and logial thinking in children and young adults, or, more generally, beginners in programming. The hardware in which the environment is based in is cheap in comparison to similar ones commercially available, and even in its abscence a virtual version of the robot can be used, version in which gamification aspects regarding the teaching of basic programming concepts were made built-in. The robot used is based on the robot from the project Jabuti Edu, from UFRGS, and was taken as starting point for this work's development. The development was guided by, and is based on, research on the scientific literature regarding educational robotics, block programming and gamification. Through the efforts described in the development, the objectives were met and the environment was made available as a webpage.

Keywords: Block programming, educational robotics, gamification.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

AGPL	Affero General Public License
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CDN	Content Delivery Network
CSS	Cascading Style Sheets
HDMI	High-Definition Multimedia Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
MEC	Ministério da Educação
MIT	Massachusetts Institute of Technology
PC	Personal Computer
PWM	Pulse Width Modulation
RPC	Remote Procedure Call
SDP	Session Description Protocol
STEM	Science, Technology, Engineering, and Mathematics
UFRGS	Universidade Federal do Rio Grande do Sul
UFSC	Universidade Federal de Santa Catarina
USB	Universal Serial Bus

SUMÁRIO

1 INTRODUÇÃO

Tanto o ensino quanto a aprendizagem de habilidades de programação são complexos e cursos de programação contam geralmente com índices altos de reprovação ou desistência (SOUZA; BATISTA; BARBOSA, 2016). Curiosamente, há relatos de que, muitas vezes, os estudantes entendem o que necessita ser programado sob um ponto de vista funcional, tendo dificuldades em efetivamente descrevê-lo conceitualmente em uma linguagem de programação (i.e. implementar um programa/aplicação) (PITEIRA; HADDAD, 2011). Este trabalho apresenta um esforço no desenvolvimento de um ferramental hardware-software com a capacidade de ser utilizado para o ensino e aprendizagem de programação.

O ambiente foi desenvolvido para uso em um robô educacional proveniente do projeto de robótica educacional Jabuti Edu, da Universidade Federal do Rio Grande do Sul, realizadas em um trabalho da disciplina de projetos de sistemas ubíquos, no Campus da UFSC - Araranguá, sendo, portanto, uma contribuição a um projeto de outra instituição. A integração entre o hardware, que é de baixo custo, e o software, acessível através de um ambiente *web*, foi motivada pelo potencial de impacto positivo em pessoas iniciantes em programação.

1.1 MOTIVAÇÃO

A habilidade de programar é uma demandada atualmente do mercado de trabalho e, ao mesmo tempo, cursos que capacitam pessoas a programar são amplamente acessíveis, assim como o próprio ferramental requerido para treinar a habilidade (como um computador pessoal e conexão à Internet). Tal habilidade é unicamente versátil em aplicações nas mais variadas áreas da indústria, e atualmente tem-se em um típico telefone celular mais poder computacional do que aquele que os astronautas tiveram quando foram à lua (CASINADER, 2019). As afirmações iniciais parecem ser mutualmente exclusivas dado que o mercado é regido por oferta e demanda, mas não o são; a habilidade de programar reflete diretamente a capacidade do raciocínio de um indivíduo, estando relacionada à capacidade de decompor problemas em módulos menores, de uso de raciocínio analítico, e de sistematicamente planejar, codificar e depurar uma aplicação. É ainda necessário que o indivíduo incorpore os conceitos de variáveis e estruturas recursivas, que são bem conhecidos por serem difíceis de serem ensinados, sendo mais facilmente entendidos no contexto de uma linguagem de programação (KURLAND et al., 1986).

Alia-se a isso a flexibilidade fornecida por linguagens de programação típicas, que são textuais, no contexto de alguém inexperiente à linguagem, isso é potencialmente negativo. Erros de sintaxe têm a capacidade de intimidar uma pessoa que esteja aprendendo a programar; tais erros, contudo, não significam incapacidade de programar. As linguagens de programação em blocos essencialmente removem esses obstáculos ao tornar impossível

a codificação de um programa sintaticamente incorreto (MERKOURIS; CHORIANOPOULOS; KAMEAS, 2017), impedindo que blocos encaixem uns nos outros caso a sequência lógica não faça sentido, o que, simultaneamente, fornece um feedback imediato, eficiente, porém sutil, ao programador a respeito de uma situação em que uma determinada sequência lógica não seja coerente. Tais linguagens vêm se tornando ferramentas essenciais para instigar crianças e jovens a programar, e ambientes de programação em blocos têm todas as características necessárias para serem usados em cursos introdutórios envolvendo programação e ciência de computação (FEDERICI, 2011).

Essas ideias motivaram o conceito e protótipo de uma plataforma integrada acessível para o ensino de conceitos básicos de programação, o trabalho aqui apresentado, que possibilita também uma comunicação com um robô educacional físico de baixo custo. A utilização deste robô educacional irá possibilitar desenvolver uma programação em bloco executar o código no robô físico ou virtualmente.

A utilização do elemento robótico, bem como aspectos de gamificação, apoia-se na teoria construtivista do conhecimento, servindo como estímulo à descoberta e exploração do ambiente integrado e, consequentemente, de elementos de lógica de programação. Estudantes são comprovadamente mais propensos a engajar-se futuramente em atividades envolvendo programação se expostos a elementos robóticos (MERKOURIS; CHORIANOPOULOS; KAMEAS, 2017).

1.2 OBJETIVOS

Nesta seção, são expostos os objetivos gerais e específicos deste trabalho.

1.2.1 Objetivo Geral

De forma geral, o objetivo do trabalho e pesquisa aqui expostos é fornecer um ambiente de programação em blocos *open-source* de baixo custo em que é oferecido aos usuários a capacidade de programar um robô baseado no robô do projeto Jabuti Edu, da UFRGS, ou uma versão virtual desse artefato robótico.

1.2.2 Objetivos Específicos

Considerando o objetivo geral apresentado e as pesquisas realizadas, destacam-se os seguintes objetivos específicos:

- Permitir ao usuário codificar um programa em blocos e executá-lo imediatamente no artefato robótico ou virtual;
- Mitigar a possível falta de acesso ao artefato robótico físico por meio da utilização do jabuti virtual;

- Disponibilizar, através da própria plataforma, simples aspectos de gamificação baseados em estruturas básicas de ciência de computação, como instruções condicionais e loops.

Ademais, como objetivo secundário, a disponibilização do código visa fomentar iniciativas similares por parte da comunidade científica.

1.3 METODOLOGIA

O trabalho documentado é um trabalho de caráter experimental baseado em pesquisas científicas e as etapas em que o desenvolvimento se deu podem ser visualizadas, em ordem cronológica, na sequência abaixo:

- Análise e definição do escopo do trabalho;
- Pesquisa na literatura científica acerca de linguagens de programação em blocos, projetos de robótica educacional envolvendo linguagens de programação em blocos e gamificação;
- Definição dos requisitos funcionais do protótipo a ser desenvolvido;
- Definição das tecnologias a serem utilizadas para o desenvolvimento do protótipo;
- Implementação do protótipo;

1.4 ORGANIZAÇÃO DO TRABALHO

Esta monografia organiza-se em cinco capítulos. O primeiro capítulo o capítulo atual, em que introduz-se a motivação por trás do trabalho e a temática em que ele está inserido e seus objetivos.

O segundo capítulo discute a fundamentação teórica em que o trabalho se baseia, em que discorre-se sobre tecnologias educacionais como linguagens e ambientes de programação utilizadas com viés educacional e robôs educacionais, a biblioteca utilizada na implementação do protótipo, a eficácia que elementos de gamificação podem ter e alguns trabalhos semelhantes desenvolvidos na década passada e na presente.

O terceiro trabalho é o cerne deste documento e expõe o ambiente integrado que foi desenvolvido, tanto conceitualmente quanto tecnicamente quanto sob a perspectiva do usuário.

O quarto capítulo apresenta os resultados obtidos através do trabalho e os discute, considerando os objetivos expostos no primeiro capítulo e a pesquisa científica realizada.

O quinto e último capítulo expõe as conclusões acerca do trabalho, evidenciando pontos passíveis de serem aprimorados em trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Serão aqui apresentados conceitos e tecnologias relevantes para o contexto do presente trabalho, segundo as pesquisas científicas e tecnológicas realizadas.

2.0.1 Tecnologias de linguagens de programação educacionais

Ao longo da história da computação, linguagens de programação educacionais e ambientes envolvendo tais linguagens têm sido desenvolvidos. No texto que segue, apresentar-se-ão alguns dos mais notáveis deles, tendo em vista o contexto deste trabalho.

2.0.1.1 LOGO

Logo é uma linguagem de programação educativa criada em 1967 por Wally Feurzeig, Seymour Papert e Cynthia Solomon. Ela foi inicialmente concebida para o ensino de matemática (FEURZEIG; LUKAS, 1972), e é uma adaptação multiparadigma da linguagem funcional LISP, sendo na maioria das implementações interpretada, embora existam implementações em que a linguagem é compilada.

Em LOGO, usando comandos textuais, controla-se um cursor, chamado tartaruga, que desenha gráficos por onde passa. Tal cursor é descrito como um "animal cibernético controlado por computador" (PAPERT, 1980). Tradicionalmente, os comandos da linguagem se referem a desenhar ou pintar, porém implementações mais modernas trazem diversas outras funcionalidades, como trabalhar com expressões matemáticas e com conceitos de inteligência artificial.

LOGO é notável por ser a primeira linguagem especificamente projetada levando em consideração o modo como as pessoas pensam (TVONTARIO, 1983), abstraindo questões técnicas de um computador, de forma análoga a como linguagens funcionais (LISP, da qual derivou-se LOGO, sendo uma delas) tendem a fundamentar-se em conceitos matemáticos, diferentemente de linguagens procedurais típicas, que geralmente surgem com base na arquitetura do hardware computacional.

A linguagem é, contudo, cognitivamente complexa e difícil de ser aprendida sem instruções, mesmo quando estudantes estão realmente motivados e engajados a aprendê-la e usá-la. A idealização de que habilidades de programação e de resolução simplesmente floresceriam ao estudantes serem expostos a ambientes de programação LOGO foi criticada, sob o argumento de que não encontrou-se evidências significativas de melhorias nas habilidades de planejamento de resolução de problemas na comparação entre dois grupos de crianças na faixa etária de 8 a 12 anos, sendo que integrantes de um dos grupos possuíam experiência de um ano de programação LOGO (PEA, 1987).

2.0.1.2 Linguagens de programação em blocos

Em LOGO, assim como na maioria das linguagens de programação, há programas inválidos; a incorreta digitação de um comando faz com que ele não seja interpretado. As linguagens de programação em blocos não permitem que erros sintáticos existam, apenas erros semânticos, o que é particularmente adequado para o ensino de lógica e conceitos de programação, até mesmo não-guiado, como é o caso do Blockly Games (ver adiante).

A diferença entre usar uma linguagem de programação em blocos e usar uma linguagem de programação textual seria equivalente a ser incapaz de errar a pronúncia de palavras, bem como ser incapaz de formar sintaticamente uma frase que não faça sentido. Nessa analogia, o texto seria o programa, e o processo de escrita do texto estaria simplificado devido a haver menos maneiras do texto ser incoerente (ele não o seria caso frases ou construções simples fossem analisadas separadamente, por design). A menor unidade passível de erro deixa de ser a palavra (ou a letra, no caso do texto ser escrito cursivamente) e passa a ser a coerência entre frases sintaticamente corretas.

Embora possa, a primeiro momento, parecer que essas linguagens não sejam tão capazes quanto linguagens de programação textuais, isso não é verdade. Embora elas possam ser propositalmente implementadas de forma limitada, em contextos específicos, a fim de facilitar a realização de tarefas que normalmente demandariam conhecimento de algum domínio técnico, como é o caso da realização de uma transação via *smart contract* (WEINGAERTNER et al., 2018), uma linguagem de programação em blocos pode ser feita tão poderosa quanto se queira. Projetos desenvolvidos pela comunidade Scratch são exemplos de como linguagens de programação em blocos não deixam de ser poderosas (GOOGLE, 2021).

A primeira linguagem do tipo foi Logo Blocks, desenvolvida pelo Massachusetts Institute of Technology (MIT), em 1996, e englobava código LOGO em blocos capazes de serem arrastados na tela e encaixados uns nos outros, compondo um programa (TEMPEL, 2013).

2.0.1.3 Scratch

Em 2003, também desenvolvido no MIT, foi criada a primeira versão do Scratch, sendo disponibilizada em 2007. Scratch é um ambiente gratuito de programação em blocos, atualmente disponibilizado através de website de mesmo nome, que permite a seus usuários criarem projetos interativos e ricos em conteúdo de mídia e disponibilizá-los à comunidade Scratch. Ele foi concebido para ser mais facilmente manipulável, mais significativo e mais social do que LOGO, sendo que inexiste algo como um passo de compilação, sendo permitido ao usuário modificar o programa enquanto ele está rodando, o que facilita a exploração do ambiente e descoberta de novos comandos (RESNICK et al., 2009). Fragmentos de programa também podem ser deixados soltos no ambiente, não

impedindo a execução do programa principal.

O ambiente foi projetado de tal forma a ter um conjunto mínimo de instruções, sendo, ainda assim, *Turing-complete*. As variáveis são visíveis em uma janela, o que as torna mais concretas ao usuário. Comandos em blocos podem atuar sobre sprites, que são desenhos SVG exibidos na tela do ambiente Scratch. Tais sprites são objetos e podem possuir estado (variáveis) e comportamento (funções) porém a linguagem não é orientada a objetos (carece de conceitos como herança e polimorfismo), mas sim baseada em objetos (MALONEY et al., 2010).

Scratch é um projeto de código aberto que oferece a possibilidade de criação de extensões com blocos customizados e capacidade de interação com hardware (e.g. o kit educacional robótico comercial LEGO MINDSTORMS EV3). Até o momento de escrita, porém, *websites* externos têm de ser usados para executar extensões desenvolvidas independentemente. Até a versão 2.0 de Scratch, fazia-se uso do descontinuado software Adobe Flash Player, sendo que a partir da versão 3.0 (atual no momento de escrita deste documento) o software foi reescrito sem necessitar de flash (usando elementos HTML5), passando a ter seus blocos com base no *framework* Blockly.

Estudos comprovam a existência de diferenças no progresso do aprendizado quando se utiliza uma linguagem de programação em blocos em comparação a quando se utiliza uma linguagem de programação textual (LOGO inclusa). Em determinado estudo, verificou-se que estudantes foram capazes de abstrair o conceito de instruções condicionais com mais facilidade ao utilizar Scratch quando em comparação com LOGO. Em contrapartida, aqueles que utilizaram LOGO sentiram-se mais confiantes com relação a suas habilidades de programação, e, interessantemente, internalizaram melhor o conceito de loops (LEWIS, 2010). Em um outro estudo (PRICE; BARNES, 2015), verificou-se, em comparação a estudantes usando linguagens de programação textuais, que os que usaram linguagens de programação em blocos demonstraram melhor desempenho nos critérios de eficiência, tempo dispendido e taxa de término de exercícios do estudo, indo de acordo à ideia de que programação em blocos podem afetar positivamente o estudo de iniciantes em programação. É também enfatizada a importância da interface dos blocos em um ambiente de programação em blocos.

2.0.1.4 Blockly

Blockly é uma biblioteca *open-source* para o lado cliente de páginas web (na linguagem *JavaScript*), disponibilizado sob a licença Apache 2.0. Ela é utilizada para criar linguagens de programação em blocos, e adiciona uma interface de usuário gráfica (*Graphical User Interface* - GUI) à tela da aplicação, com um ambiente de blocos à disposição do usuário.

Um projeto do Google lançado em 2012, Blockly é, sob a perspectiva do programador, um meio através do qual o usuário de sua aplicação "digita" código sintaticamente

correto (GOOGLE, 2021) em JavaScript, Python, PHP, Lua, Dart ou XML, sendo possível também implementar diferentes linguagens de saída.

Blockly não é nem uma aplicação nem uma linguagem, e sim uma ferramenta usada por diversos projetos (notavelmente Scratch, citado acima) com a qual se projeta uma linguagem, tendo em vista os requisitos da aplicação sendo desenvolvida (PASTERNAK; FENICHEL; MARSHALL, 2017).

O design de uma linguagem em blocos é de suma importância tendo em vista a capacidade da linguagem influenciar o processo cognitivo que o usuário terá. Os blocos providos pelo Blockly são customizáveis ao ponto de ser possível criar uma linguagem em que muitas coisas desnecessárias foram adicionadas, que apresente blocos redundantes ou que não seja facilmente entendida pelos usuários, o que acarretaria em uma experiência negativa ao usá-la.

2.1 GAMIFICAÇÃO

Embora as linguagens de programação em blocos obtenham certo sucesso ao remover o elemento frustração do processo de aprendizagem, não se está adicionando elementos motivacionais no processo, apenas removendo-se elementos possivelmente desmotivacionais. Uma proposta mais ambiciosa, mas, ao mesmo tempo, desafiadora, é remover tal neutralidade do ambiente de ensino-aprendizagem, de maneira a torná-lo intrinsecamente motivacional. Para tal, é desejável que ele forneça entretenimento ao usuário.

Ao uso de mecânicas e estéticas típicas de jogos para promover o engajamento em uma atividade denomina-se gamificação (KAPP, 2012). Não deve-se confundi-la, porém; o propósito de um processo gamificado não é promover entretenimento, e sim adicionar elementos característicos de jogos como barras de progresso, pontos de experiência, desafios/atividades, recompensas (como medalhas), rankings, ou mesmo itens, secundariamente ao propósito original.

A abordagem mostra-se efetiva em variados contextos. No âmbito corporativo, estratégias de gamificação fizeram com que a habilidade de aprendizado fosse aprimorada em 40% (GIANG, 2013). Uma abordagem gamificada aplicada à dieta escolar de uma escola estadunidense foi capaz de aumentar o consumo de frutas em 66% e de vegetais em 44% em um período de apenas 13 dias (JONES et al., 2014). O sucesso de Duolingo e Memrise, que gamificam o aprendizado de linguagens naturais, sugere que o aprendizado linguístico pode também beneficiar-se da gamificação, como é o caso do *website* CodeAcademy, uma plataforma de aprendizado eletrônico gamificada que possui um *dashboard* apresentando todo o conteúdo de determinado curso, possuindo também conquistas obtidas e medalhas de recompensa.

Em um contexto educacional mais genérico, há plataformas, como o Moodle, que oferecem funcionalidades com viés gamificado, tais quais fotos de perfil ou avatares para os usuários, visibilidade do progresso dos estudantes em formato de barra de progresso,

resultados de quizzes que mensuram o nível do conhecimento adquirido e potencialmente os exibe em formato de ranking customizável (adicionando, assim, um elemento de competitividade à atividade educacional, elemento esse muito presente em jogos), sistema de níveis (levels) e pontos de experiência, sistema de feedback, medalhas e ranking de líderes, bem como a possibilidade de exibição de determinadas atividades apenas caso determinada condição, em função de elementos do ambiente, seja verdadeira (o que remete à ideia de "desbloquear" determinada atividade ao completar uma outra, ou cumprir com determinado critério) (KIRYAKOVA; ANGELOVA; YORDANOVA, 2014).

Um exemplo de gamificação no contexto de linguagens de programação em blocos é Blockly Games, que consiste em uma série de jogos educativos usando Blockly, disponibilizada pela Google através de um *website*, sendo os jogos projetados para dispensarem um professor ou instrutor (FRASER, 2015).

De maneira complementar aos benefícios inerentes às linguagens de programação em blocos, como a incapacidade de cometer-se um erro sintático, é possível fazê-las mais cativantes para determinado público ao adicionar elementos de gamificação que atraiam tal público. Um exemplo bem-sucedido disso é *Storytelling Alice*, um ambiente de programação baseado no ambiente Alice, que introduz garotas estadunidenses (cujas faixas etárias estariam como aquelas das garotas nos anos finais do ensino fundamental, em relação ao Brasil) à programação colocando como objetivo criar histórias tridimensionais animadas. Embora, comparado à versão genérica de Alice, não houve diferenças significativas na taxa de aprendizado dos conceitos de programação, as garotas que usaram o Storytelling Alice demonstraram 42% mais motivação ao fazê-lo em comparação às que usaram o ambiente Alice convencional (KELLEHER; PAUSCH; KIESLER, 2007). O estudo citado sugere que levar em consideração interesses em comum de determinado grupo de estudo, ou mesmo de um indivíduo, pode fazer com que os indivíduos sintam-se mais motivados a aprender os conceitos de programação.

Sob uma perspectiva mais abstrata, pode-se argumentar que um ambiente educacional gamificado, na hipótese de ser bem-sucedido no intento do estudante adquirir determinado conteúdo, são benéficos tanto para o instrutor ou instituição de ensino, que deseja que o estudante adquira o conteúdo, quanto para o próprio estudante, pois além de estar aprendendo estará se divertindo.

Em suma, a gamificação mostra-se como uma estratégia efetiva para ser usada em um ambiente educacional, sendo capaz de trazer benefícios para um ambiente de aprendizado, quando utilizada de forma a instigar uma competição saudável e amigável, gerando motivação adicional aos estudantes. Ela possui efeito positivamente duplo: ao mesmo tempo que afeta o desempenho e entendimento dos estudantes com relação a determinado conteúdo, também cria condições para um processo de aprendizado melhor (KIRYAKOVA; ANGELOVA; YORDANOVA, 2014).

2.2 ROBÓTICA EDUCACIONAL

Ao longo da última década, o uso da robótica para fins educacionais tem despertado interesses por parte de professores e pesquisadores como uma ferramenta valiosa para o desenvolvimento de não apenas habilidades cognitivas, mas também sociais, em estudantes de variadas idades. Embora essa área, que está em crescimento, possa ser utilizada para qualquer objeto de estudo, ela é vista como sendo capaz de significativamente impactar a educação científica e tecnológica, desde o ensino básico até o universitário (ALIMISIS, 2013).

Há diversas iniciativas de usos de robôs para a chamada educação Science, Technology, Engineering, and Mathematics (STEM) e o conceito pode ser incorporado em praticamente todas as áreas, inclusive as artes (CROSS et al., 2013). Convenientemente, o uso da robótica educacional em quaisquer áreas faz com que estudantes tendam a desenvolver aptidão por conceitos de engenharia, física e mecânica, até mesmo em estudantes que não mostram interesse imediato em tais áreas (CROSS et al., 2013), possibilitando um ensino multidisciplinar integrando aspectos técnicos e sociais das disciplinas.

Em acordo à teoria construtivista do conhecimento, em um ambiente em que robótica educacional esteja sendo utilizada o papel do instrutor passaria de ser o tradicional, de detentor final do conhecimento, para o de mediador entre a aquisição de conhecimento obtida pela exploração por parte dos estudantes (BILOTTA et al., 2009).

O uso desse artifício na educação deve ser sensato, e o fato dele estar sendo usado não pode ser algo mais importante do que o motivo pelo qual ele está sendo usado. Diferentemente de quando funciona como complemento educacional, a substituição de um método educacional tradicional por um que faça uso de robótica idealmente não deve impor obstáculos ou oferecer alguma espécie de impedimento a algo que anteriormente estava sendo ensinado. Foi o caso em um estudo em que, ao longo de um ano, resultados de prova de 800 estudantes de ciência de computação foram comparados, sendo que parte dos estudantes havia tido aulas de laboratório com uso de robótica e outra parte, sem (FAGIN; MERKLE, 2003). Apesar das provas serem idênticas, o resultado do estudo foi negativo; os estudantes submetidos a robótica educacional acabaram se saindo pior que os que não foram submetidos, e isso foi dado em parte pelos estudantes poderem apenas rodar e depurar seus programas durante o uso do laboratório, não tendo fácil acesso a fazê-lo fora dele.

Um obstáculo para a difusão da robótica educacional é o possível custo elevado do hardware. A exemplo, o popular kit comercial Lego Mindstorms, particularmente o Ev3, que é projetado especificamente para uso educacional, custa, no momento de escrita deste documento, aproximadamente dez vezes mais que o custo total do hardware em que o trabalho presente se baseia.

2.3 TRABALHOS RELACIONADOS

A seguir, serão mencionados alguns trabalhos relacionados ao presente trabalho no âmbito nacional e internacional.

2.3.1 BlockC

Em 2011, uma implementação em blocos mínima da linguagem de programação C, notavelmente conhecida por ser um dos pilares da computação, foi feita utilizando uma versão modificada do então software Scratch para desktops (BYOB mod) por Stefano Federici, do Departamento de Educação e Filosofia da Universidade de Cagliari - Itália.

Certas funcionalidades foram suprimidas, como manipulação de ponteiros, structs e manipulação de arquivos, no entanto a implementação manteve-se fidedigna às keywords, tipos numéricos (limitados a *int*, *float* e *char*, e *arrays* limitados a unidimensionais e bidimensionais) e até mesmo a funções que caracterizam entrada e saída de dados em C, como a printf e a scanf. O intuito primordial dos esforços de Federici foi permitir uma transição suave de BlockC a ambientes de desenvolvimento C padrão, uma vez que o maior obstáculo à introdução dos estudantes à linguagem de programação C é sua sintaxe (FEDERICI, 2011).

Federici cita, contudo, como limitação a seu trabalho, a incapacidade da modificação utilizada (BYOB) esconder alguns aspectos visuais de sua utilização, bem como a possibilidade do usuário apagar arquivos essenciais para a correta utilização de BlockC, os quais BYOB usa. O código de BlockC é aberto.

Scratch, no momento de escrita deste documento, é capaz de rodar através de um navegador *web* (não o era) e provê a capacidade de codificar extensões que alteram ou adicionam blocos ou comportamento de blocos.

2.3.2 LearnBlock

Em 2020, um trabalho desenvolvido no Laboratório de Robótica e Visão Artificial (Robotics and Artificial Vision Laboratory - RoboLab) do Departamento de Tecnologia de Computação e Comunicação (Department of Computer and Communication Technology) da Universidade de Estremadura - Espanha, que foi apoiado em parte pelo governo da Espanha e também pelo de Estremadura, apresentou LearnBlock, cuja principal propriedade é descrita como sendo agnóstica em termos do robô utilizado (BACHILLER-BURGOS et al., 2020).

LearnBlock usa a biblioteca Blockly para gerar código Python capaz de interagir com os robôs que controla. Isso é obtido através da definição de classes Python capazes de interagir com os diferentes dispositivos providos pelo robô, bem como métodos de conexão e desconexão. O usuário pode escolher entre executar o código em blocos, o código Python

gerado através do código em blocos ou diretamente codificar e executar código Python, sendo todas as três opções independentes do hardware.

Os blocos exibidos pela interface gráfica de LearnBlock são customizáveis através de uma linguagem declarativa específica chamada Block-Text. O projeto do qual LearnBlock emergiu também desenvolveu um robô, o EBO, e ambos (tanto o LearnBlock quanto o EBO) são open-source.

A arquitetura é tal que abstrai o hardware; a funcionalidade de exibir uma "emoção" em um robô, por exemplo, é emitida por LearnBlock da mesma forma, mas, para um robô, pode culminar na modificação de um display, enquanto que, em outro, pode culminar no acendimento de LEDs.

O trabalho cita que, embora LearnBlock seja desenvolvido como aplicação desktop, primariamente devido à possibilidade do requerimento da instalação de alguns softwares específicos por parte de robôs, é indubitável que ferramentas educacionais baseadas na *web* são mais acessíveis por dispensarem quaisquer tipos de instalação. Enfatiza ainda que a maior limitação de ferramentas educacionais web estaria na execução do código, devido à comunicação entre robôs distintos ser distinta, o que estaria em desencontro com a ideia principal do projeto (agnóstico em termos de robô).

2.3.3 KareNao

Em 2021, em trabalho desenvolvido pela Universidade de Auckland - Nova Zelândia e submetido à décima oitava conferência internacional de robôs ubíquos (International Conference of Ubiquitous Robots), foi apresentada uma implementação de um sistema tangível baseado em blocos (*tangible block-based system*) intitulado KareNao (FU et al., 2021). Baseado no conceito de TUIs (tangible user interfaces) em oposição a GUIs (*graphical user interfaces*), "kare" significa "amigo" em maori (Nao advém do fato de que o sistema usa um robô chamado Nao), e o nome faz alusão ao ambiente amigável que a abordagem teria para o aprendizado de crianças.

O sistema, usando uma biblioteca JavaScript de código aberto desenvolvida pela Universidade de Tufts (Medford, Massachusetts), reconhece marcas fiduciais de blocos físicos utilizados para a programação através de uma câmera e as interpreta logicamente compondo código, o que ultimamente possibilita crianças programarem utilizando objetos físicos representando os blocos ao invés de interagirem com um dispositivo computacional convencional (com mouse ou touchscreen).

Os blocos físicos representando funções básicas oferecidas pelo robô Nao encaixam-se uns nos outros e seu arranjo representa lógica digital, assim como seus correspondentes virtuais (também existentes). As crianças apresentaram menos dúvidas durante a utilização dos componentes tangíveis em comparação aos virtuais. O trabalho cita as crianças sentirem-se intrigadas com os blocos, tentando organizá-los de várias maneiras diferentes e aguardando feedback de como o programa comportar-se-ia. Esse comportamento é con-

dizente com a teoria construtivista do conhecimento, e, metaforicamente, pode-se pensar nos blocos como sendo peças de um quebra-cabeça, e a solução para o quebra cabeça não estaria bem definida (sendo definida pelo usuário), mas pode ser entendida como os movimentos que o usuário gostaria que o robô fizesse.

Notou-se que era necessário fornecer mais instruções para as crianças serem capazes de utilizar os blocos virtuais em comparação aos físicos; para os físicos, bastou apresentá-las onde posicionar os blocos e como executar o programa, enquanto, para os virtuais, questões como onde encontrar os blocos e como arrastá-los para a tela tinham de ser descritas. Em contrapartida, há um número finito de blocos físicos que podem ser mostrados simultaneamente à câmera, o que limita o tamanho que um programa pode ter.

2.3.4 BetaBot

Em 2014, um trabalho da Universidade do Oeste Paulista propôs BetaBot (MAIA et al., 2014), um conjunto hardware-software composto de um robô controlado por uma placa computacional Raspberry P e um microcontrolador Arduino. O intuito do trabalho foi apoiar o ensino de computação e robótica, sendo o robô controlado por código Python e possuindo funcionalidades simples de visão computacional, como a capacidade de detecção de figuras geométricas simples através de uma webcam. O dispositivo robótico também apresenta alguns sensores como um de distância, porém não faz uso de programação em blocos.

2.3.5 EduBot

Em 2019, em trabalho desenvolvido pela Universidade Amity - Índia, foi apresentado EduBot (ISLAM et al., 2019), um conjunto hardware-software desenvolvido com foco em crianças não-privilegiadas da região pacífica da Ásia. O design foi feito de maneira a ser econômico, sendo que a placa de circuito impresso foi concebida especificamente para tal, fazendo uso de um microcontrolador ATmega328 e destina-se a pessoas de quaisquer idades interessadas em aprender lógica e conceitos básicos de programação usando seus smartphones. Foram também incorporados à placa de circuito impresso sensores como de temperatura, de distância e de detecção de gás.

Para controlar o robô, foi usada programação em blocos através da biblioteca *open-source* Blockly em uma aplicação Android nativa, nas linguagens Inglês e Bengali, sendo que os blocos foram divididos em diferentes níveis de complexidade (levando em consideração os conceitos de programação que representam).

2.3.6 Projeto Jabuti Edu

O projeto Jabuti Edu, no qual o presente trabalho se baseia, é uma iniciativa de robótica educacional brasileira idealizado por Eloir José Rockenbach e desenvolvido

pela Universidade Federal do Rio Grande do Sul, através da qual atividades educativas em diversos níveis de ensino estão sendo desenvolvidas em diferentes estados brasileiros (SILVA; ROCKENBACK; FAGUNDES, 2021).

A comunidade por trás do projeto organiza-se em núcleos, sendo que um núcleo, composto por pessoas próximas geograficamente, é responsável por ações do projeto em sua região (ALVES E.; ALVES, 2017).

O robô é em formato de jabuti, originalmente montado a partir de impressora 3D. O hardware faz uso de uma placa Raspberry Pi, com um par de motores e de rodas. Ambos software e hardware são abertos, sendo o software sob a licença Affero General Public License (AGPL) e o hardware, sob a licença CERN V1.2L. Há também o Jabuti Edu Mini, que, por sua vez, é baseado no microcontrolador Arduino.

O software para o projeto foi concebido em módulos, sendo que no primeiro módulo há botões em forma de setas que diretamente controlam o robô, além de um campo onde se pode digitar texto que será falado pelo robô (foi adaptado com uma caixa de som). No segundo, é possível codificar em LOGO e solicitar a execução do programa, que é autorizada (ou não) pelo mediador da utilização do robô.

Em trabalho desenvolvido por pesquisadores da Universidade Federal do Oeste do Pará, foi desenvolvido um novo módulo para o software do jabuti envolvendo programação em blocos através da biblioteca Blockly (ALVES E.; ALVES, 2017).

2.3.7 ROBO+EDU

Outra iniciativa da Universidade Federal do Rio Grande do Sul (UFRGS), em colaboração com o Programa Mais Educação, do Ministério da Educação (MEC), o ROBO+EDU é um projeto que visa a capacitação e formação inicial e continuada de professores e profissionais da Educação Básica que atuam em escolas ou sistemas de educação públicos, sendo esses treinados a usar um kit hardware-software desenvolvido pelo projeto. O hardware faz uso do microcontrolador Arduino, e possui sensores capazes de medir temperatura, luminosidade e som.

O design do robô foi feito com plástico injetável por ser mais resistente. O software é programado através da própria interface de programação do Arduino, ou de uma versão traduzida do ArduBlock, um ambiente de programação em blocos para Arduino. É também apontada como uma das metas do projeto instigar em crianças e adolescentes interesse pelas áreas de engenharia e tecnologia (UFRGS, 2015).

3 DESENVOLVIMENTO

Nesta seção serão apresentados os aspectos envolvidos no desenvolvimento do ambiente integrado que este trabalho propõe, tanto conceitualmente quanto sob um ponto de vista técnico. Serão também apresentados diagramas ilustrando e exemplificando o ambiente e seu uso sob a perspectiva do estudante e do professor/instrutor.

3.1 REQUISITOS FUNCIONAIS

No intuito de cumprir os objetivos propostos, foram estabelecidos os seguintes requisitos funcionais para o ambiente desenvolvido:

- O usuário deve ser capaz de codificar um programa usando uma linguagem de programação em blocos;
- O usuário deve ser capaz de decidir entre usar o jabuti físico ou sua versão virtual;
- O usuário deve ser capaz de executar código bloco a bloco;
- A execução do programa deve ser evidenciada através de uma identificação visual do bloco sendo executado;
- Deve haver um mapa de blocos e um jabuti virtual integrados ao próprio ambiente;
- O usuário deve ser capaz de parar a execução de um programa a qualquer momento;
- O usuário deve ser capaz de iniciar um jogo;
- Deve haver diferentes níveis de jogo, cada qual com uma dificuldade distinta, gradativamente aumentando com o nível;
- Deverá existir um desafio de programação que necessariamente exija o uso do conceito de estruturas de repetição;
- Deverá existir um desafio de programação que necessariamente exija o uso do conceito de instruções condicionais.

Devido à assimetria das rodas do *hardware* utilizado, também foi estabelecido o seguinte requisito funcional:

- Deve ser possível a um administrador configurar os movimentos físicos do jabuti

3.2 USABILIDADE

Nesta seção será apresentada a usabilidade do sistema sob as perspectivas do usuário e do instrutor.

3.2.1 Perspectiva do usuário

Sob a perspectiva do usuário, o ambiente integrado desenvolvido é através do acesso a um *website*, sendo, portanto, necessária uma conexão à Internet. É possível também, porém, caso se faça necessário, utilizar o ambiente localmente sem a necessidade de uma conexão à Internet obtendo uma cópia local do repositório *git* de desenvolvimento do ambiente integrado e ao executar o arquivo entitulado *index.html* em um navegador.

Acessando o ambiente, o usuário terá à sua disposição um ambiente de programação em blocos com o qual pode codificar um programa. Terá também um botão que o possibilitará executar o programa, sendo que as instruções envolvendo os movimentos serão refletidas no jabuti que estiver selecionado no momento da execução do código (ou o físico ou o virtual).

O usuário poderá alternar entre o uso do jabuti físico e do virtual através de uma caixa seletora, conforme demonstra figura 7. Também poderá pausar a execução de um programa através do pressionar de um botão. Caso a *checkbox* "Executar bloco a bloco"esteja marcada, o programa será automaticamente pausado após cada execução de bloco.

Além disso, o que diz respeito aos controles relativos às gamificações, há botões para dar início aos desafios implementados no ambiente, e um botão para inicializar uma gamificação normal (sendo que o nível da gamificação será o selecionado).

A programação é feita através do *workspace*, conforme nomenclatura do Blockly, e o usuário dispõe blocos na intenção de formar um programa. Os blocos são obtidos através da seleção de uma categoria de blocos e dos subsequentes arrastos deles para compor o programa. Em circunstância que o usuário desejar remover bloco(s), ele pode fazê-lo jogando-os na lixeira existente no canto inferior direito do *workspace*.

3.2.2 Perspectiva do instrutor

O professor ou instrutor, para utilizar o ferramental, deve conectar o jabuti físico ao mesmo ponto de acesso que o dispositivo em que o ambiente integrado está (na mesma rede local). Para tal, ele precisa conectar um cabo High-Definition Multimedia Interface (HDMI) à saída de vídeo do Raspberry Pi e conectar a outra saída do cabo em algum dispositivo capaz de reproduzir a tela.

Usando o sistema operacional do Raspberry Pi chamado Raspbian, ele poderá então conectar o Raspberry Pi ao ponto de acesso através da conectividade WiFi (protocolo IEEE 802.11) provida pelo Raspberry Pi. O sistema operacional engarrega-se de dispensar o mesmo processo nos usos subsequentes do sistema, sendo necessário refazer o processo, porém, caso o jabuti seja movido para um local com uma diferente rede local.

Uma vez que o Raspberry Pi estiver conectado à rede local, o executável do *firmware* deverá ser executado. A Application Programming Interface (API) para controle do jabuti

físico estará então exposta à rede local e o ambiente integrado ou a biblioteca *jabutixlib*, em execução em outro dispositivo da rede local, será capaz de identificar o jabuti físico na rede e interagir com ele. Para que não seja necessário executar o *firmware* cada vez que o Raspbian reinicializar, é possível configurar o sistema operacional para executar automaticamente o executável do *firmware* após a inicialização.

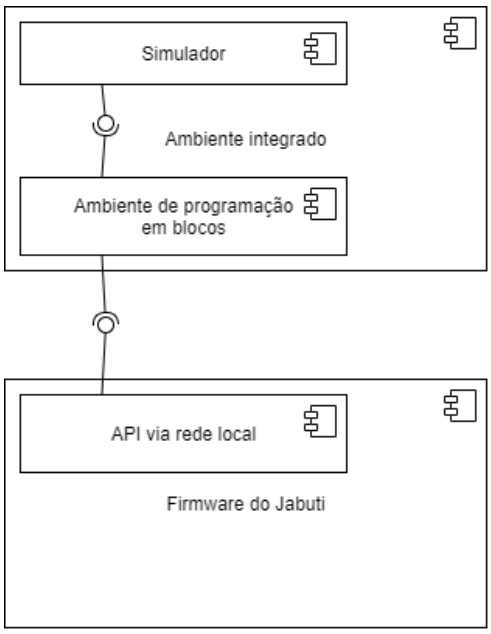
Para acessar a página administrativa, em que se pode configurar a movimentação do jabuti físico, o instrutor poderá acessar, através de um navegador em um dispositivo conectado à rede local, a página de URL:*raspberrypi:3000/admin*. Através dela, é possível configurar individualmente a potência, em termos percentuais em relação à máxima, que será aplicada às rodas do jabuti físico para os três tipos diferentes de movimentos que ele pode executar.

3.3 ARQUITETURA

A maioria dos componentes estão concentrados no ambiente integrado, com o qual o usuário interage. É crucial à arquitetura, porém, o *firmware* existente no jabuti.

Conforme será detalhadamente descrito nas seções que seguem, o código desenvolvido pelo usuário através do ambiente integrado interage com o *firmware* através de mensagens enviadas pela rede local, ou com a versão virtual do jabuti (que se encontra no próprio ambiente integrado).

Figura 1 – Diagrama de componentes da arquitetura.



Fonte: Elaborado pelo autor (2022).

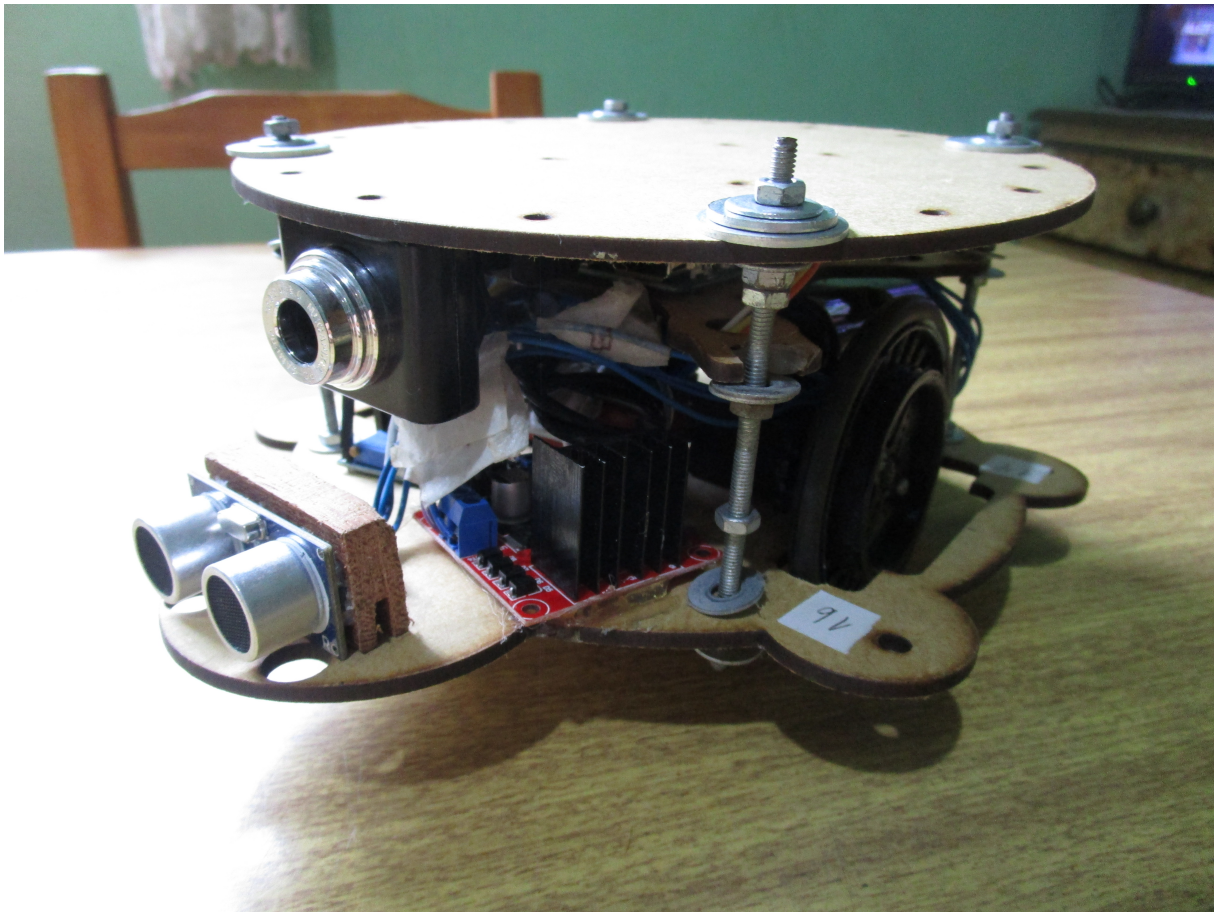
A figura 1 ilustra o relacionamento entre os componentes da aplicação, sob uma visão de alto nível. A aplicação concentra-se na página *web* que pode ser vista nas figuras

7 e 8, e os demais componenentes são componentes de software com os quais esse elemento central (pode-se pensar nele como a página *web*) interage.

3.3.1 Hardware

O *hardware* utilizado para o desenvolvimento deste trabalho, visto na figura 2, é baseado no robô do projeto Jabuti Edu, da UFRGS.

Figura 2 – Robô utilizado.



Fonte: Elaborado pelo autor (2022).

É importante enfatizar que o *hardware* utilizado não foi desenvolvido neste trabalho; conforme descrito na introdução, ele é fruto de modificações no robô do projeto Jabuti Edu feitas em um trabalho da disciplina acadêmica de projetos de sistemas ubíquos, no Campus da UFSC - Araranguá, que incorporaram no dispositivo uma câmera Universal Serial Bus (USB), um sensor ultrassônico e uma arranjo de ponte H para o acionamento dos motores. O *hardware* utilizado é uma instância dentre os possíveis *hardwares* que podem ser utilizados em conjunto com os objetos desenvolvidos no presente trabalho, sendo possível utilizar outro robô controlado por Raspberry Pi desde que o controle às

rodas seja similar (controlado através das tensões elétricas emitidas em dois pinos da placa).

O trabalho desenvolvido requer um robô que possua duas rodas controladas por pinos Pulse Width Modulation (PWM) de canais independentes independentes de uma placa computacional *Raspberry Pi*, bem como uma câmera conectada à placa via interface USB. O robô utilizado é controlado por uma placa do modelo *Raspberry Pi* 3 B+.

3.3.2 Firmware

O propósito do *firmware* é prover uma API através da qual pode-se programaticamente controlar o jabuti físico. Tal interface de programação visa possibilitar que mensagens de controle sejam enviadas por dispositivos que estejam conectados à mesma rede local do jabuti físico através de um protocolo de comunicação bidirecional. O jabuti físico, ao deparar-se com uma mensagem de controle, a executa e, possivelmente, envia de volta ao dispositivo controlador uma mensagem de *feedback*. O *firmware* também é responsável por enviar imagens da câmera desde o jabuti físico ao ambiente integrado.

Uma particularidade é que o jabuti físico desconhece o programa sendo executado, respondendo apenas a comandos em um nível mais baixo (em comparação ao de um programa completo). Em contraste, há arquiteturas cuja abordagem é oposta, todo o programa codificado pelo usuário seria enviado ao jabuti, que o interpretaria. A abordagem utilizada trata o jabuti como um mero seguidor de comandos.

Independe a origem dos comandos; o *firmware* apenas recebe o comando e o executa. Essa independência possibilita que a mesma interface de programação possa ser utilizada tanto pelo ambiente integrado quanto pela biblioteca *Python*, sendo, portanto, uma anagnosticidade desejável.

Firmwares são *softwares* especificamente projetados para uma aplicação, comumente encontrados em sistemas embarcados e tipicamente escritos em linguagens de programação mais *backbone*, como C, C++ e *Assembly*. Entretanto, como está-se fazendo uso de um *hardware* relativamente poderoso, optou-se pelo uso da linguagem *Go*, sendo o *firmware* um executável gerado através da compilação do código *Go* com funcionalidades de abstração do hardware e exposição de seu controle para uso pelo ambiente integrado.

3.4 COMUNICAÇÃO ENTRE O AMBIENTE INTEGRADO E O JABUTI FÍSICO

O *firmware* essencialmente faz com que o *hardware* do jabuti físico aja como um servidor *web*. No protótipo desenvolvido, ele estabelece um servidor HTTP na porta 3000 e, no momento em que um dispositivo conecta-se, eleva tal conexão ao protocolo bidirecional *WebSockets*. Uma conexão via protocolo *WebRTC* é também estabelecida, a partir da conexão *WebSockets* primeiramente estabelecida, com a finalidade de transmitir um *stream* de vídeo da câmera frontal do jabuti físico.

A partir do momento que a conexão *WebSockets* é estabelecida, ela é utilizada para estabelecer uma conexão *WebRTC* para fins de transmissão da câmera desde o jabuti físico ao ambiente integrado. O *WebRTC* é um projeto gratuito e de código aberto que possibilita transmissão de mídia (áudio, vídeo ou dados) em tempo real para uso em aplicações nativas *web*, *mobile* e de Internet of Things (IoT), sendo suas funcionalidades nativas em navegadores modernos.

Para que a conexão *WebRTC* seja estabelecida, uma mensagem de oferta de sessão *WebRTC*, no protocolo Session Description Protocol (SDP), é enviada a partir do ambiente integrado até o *firmware*, que recebe a mensagem e gera uma mensagem de resposta SDP correspondente. A mensagem de resposta é então enviada novamente ao ambiente integrado e o ambiente integrado. Após esse processo, uma conexão remota *WebRTC* é estabelecida a partir dos dados de sessão trocados.

Para as funcionalidades de conectividade de rede no contexto do *firmware*, foi utilizado o pacote padrão de Hypertext Transfer Protocol (HTTP) da linguagem *Go*, juntamente com os pacotes *gorilla/websocket* e *pion/webrtc*. Para interagir com a câmera, foram usadas funcionalidades providas pelo pacote *pion/mediadevices*. Para interagir com a placa *Rasberry Pi*, foi usado o pacote *stianeikeland/go-rpio*.

Para as funcionalidades de rede no contexto do ambiente integrado, foram utilizadas as funcionalidades nativas da linguagem JavaScript (*WebSockets* e *WebRTC*).

A conectividade é mais facilmente visualizada ao ser separada em duas partes: antes da obtenção, por parte do navegador, do sistema integrado e depois da obtenção. Uma vez que a página do sistema integrado entra em execução, iniciam-se as conexões acima mencionadas. O sistema integrado pode, porém, ser obtido pelo navegador através de três maneiras distintas: através de uma *CDN*, através da execução local ou, apesar de não estar ainda implementado no momento de escrita deste documento, através do fornecimento a partir do Raspberry Pi.

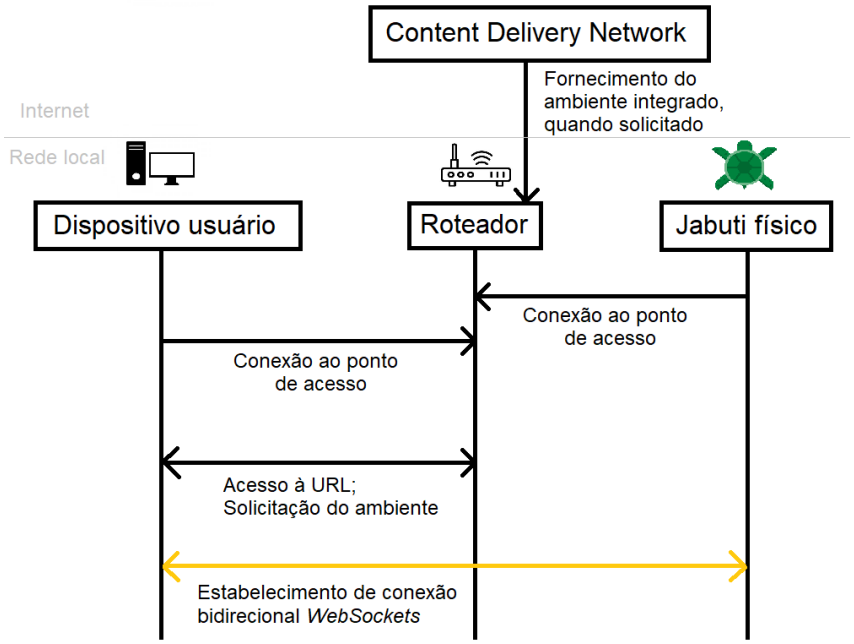
O primeiro de acesso ao ambiente integrado é através de uma *CDN*, caso em que o usuário acessa uma *URL* e os arquivos estáticos que compõem o ambiente integrado são fornecidos através de um serviço de terceiros. Esse modo de acesso ao ambiente integrado tem como vantagem ser amplamente acessível e simples, porém requer uma conexão à Internet.

O segundo modo de acesso é através da abertura da página *web* em disco ou em armazenamento removível, sendo útil para casos em que não se tem nem conexão à Internet nem acesso ao jabuti físico.

O terceiro modo de acesso seria através do fornecimento da página *web* pelo próprio jabuti físico, através de um servidor *HTTP*. Esse modo de acesso possibilitaria o acesso ao ambiente integrado através do acesso a uma *URL* e não requereria uma conexão à Internet, mas requereria que o jabuti estivesse acessível através da rede.

Estando o ambiente integrado carregado pelo navegador, ele instruirá o navegador

Figura 3 – Diagrama ilustrando o estabelecimento da conexão bidirecional com o jabuti físico a partir do acesso ao ambiente integrado fornecido por uma CDN.

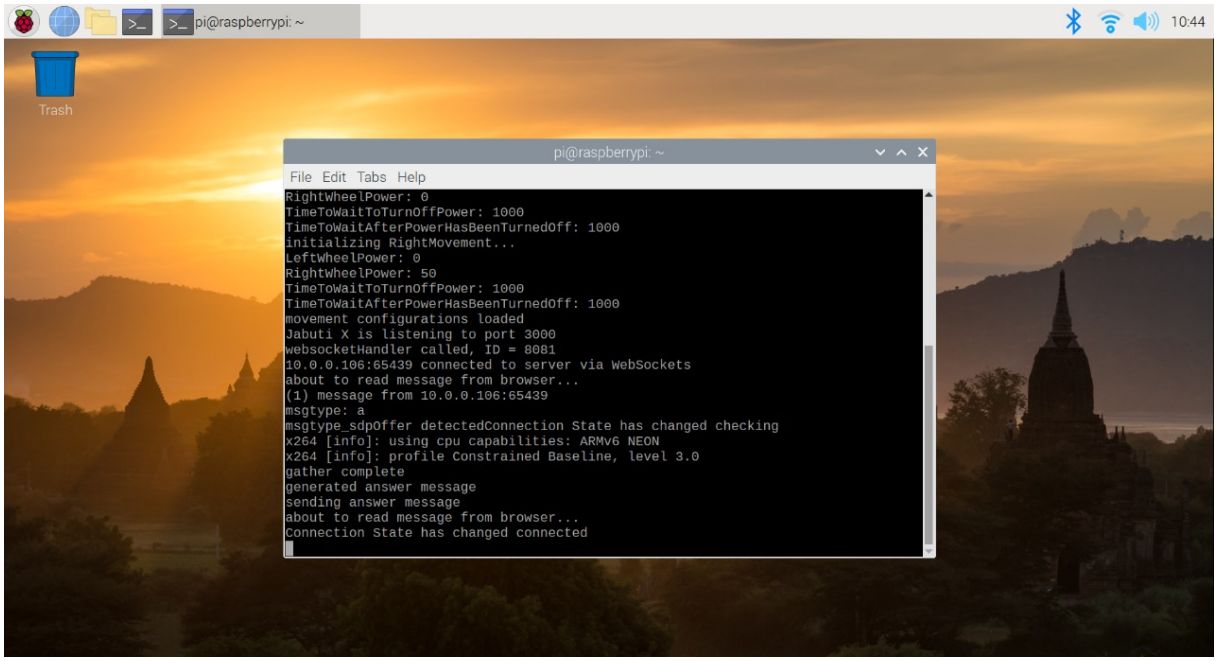


Fonte: Elaborado pelo autor (2022).

a estabelecer a conexão bidirecional (via *WebSockets*) ao jabuti físico, através da qual as mensagens de comunicação (descritas no quadro 1) são trocadas. Para que a conexão ocorra, porém, ambos o jabuti físico e o dispositivo usuário devem previamente ter sido conectados ao mesmo ponto de acesso (o jabuti físico deve estar alcançável através da rede local.

A figura 3 ilustra a interação entre os elementos de rede no contexto do uso do ambiente integrado através de uma CDN. É válido ressaltar que trata-se de uma representação em alto nível evidenciando aspectos relevantes à conectividade descrita nessa seção, e que todas as interações citadas acontecem bidirecionalmente (e.g. o *handshake* ao estabelecer conexão com a rede WiFi). As duas primeiras setas horizontais no diagrama 3 ilustra ambos o dispositivo usuário e o jabuti físico conectarem-se ao mesmo ponto de acesso. A terceira seta representa a requisição do ambiente integrado desde o dispositivo usuário, que pode ser um Personal Computer (PC) ou um telefone celular, até o serviço CDN, usando a Internet, e o subsequente recebimento deste. O ambiente integrado é então interpretado e executado pelo navegador, e estabelece uma conexão bidirecional *WebSockets* com o jabuti físico, representada pela seta bidirecional amarela.

Figura 4 – Protótipo do *firmware* desenvolvido rodando em um sistema operacional *Raspbian*.



Fonte: Elaborado pelo autor (2022).

A figura 4 ilustra o *firmware* rodando no *Raspbian*, bem como mensagens de texto emitidas pelo *firmware* relacionadas a eventos envolvendo o ambiente integrado e o jabuti físico.

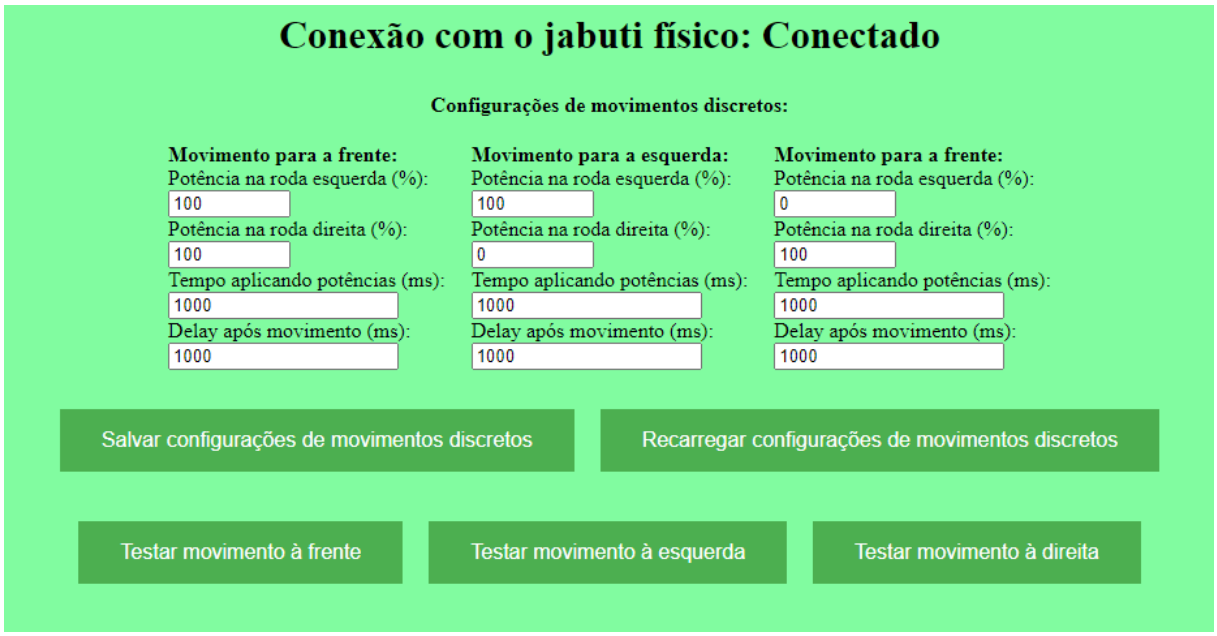
3.4.0.1 Movimentação do jabuti físico

O jabuti físico possui duas rodas simétricas que são controladas pela tensão elétrica em dois pinos da placa Raspberry Pi. Tendo em vista que as rodas podem apenas acelerar para a frente e com o intuito de padronizar a movimentação do jabuti, bem como deixá-la evidente, os movimentos foram discretizados em unidades bem definidas.

Foi convencionado que o dispositivo pode mover-se de três maneiras distintas: discretamente em uma unidade para a diagonal à esquerda de sua frente, discretamente em uma unidade para a sua frente e discretamente em uma unidade para a diagonal à direita de sua frente. Um movimento discreto consiste na aplicação de ondas elétricas quadradas moduladas em pulsos (técnica de modulação de amplitude de pulso, ou PWM) aos pinos que controlam as rodas durante algum tempo, e na subsequente não-aplicação de tensão elétrica alguma aos mesmos pinos durante mais algum outro tempo.

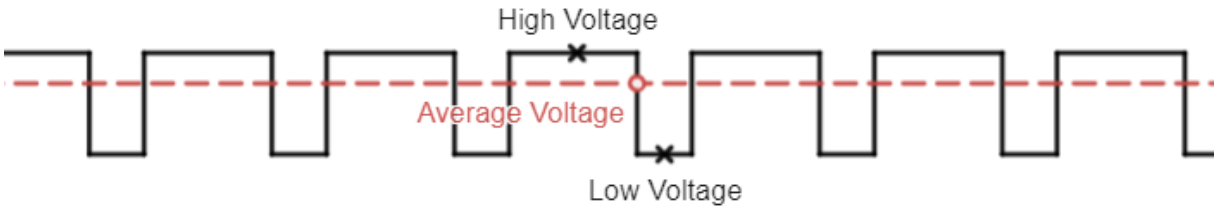
O *firmware* provê uma página *web* de configuração para tais parâmetros, acessível através da rede local em *raspberrypi:3000/admin* e ilustrada na figura 5. Essa decisão de design adveio da constatação de que as rodas do jabuti disponível para o desenvolvimento do projeto eram ligeiramente assimétricas.

Figura 5 – Página *web* para a configuração dos movimentos discretos do jabuti físico.



Fonte: Elaborado pelo autor (2022).

Figura 6 – Ilustração da técnica de modulação de amplitude de pulso, com *duty cycle* de 70%.



Fonte: Elaborado pelo autor (2022).

A técnica de modulação de amplitude de pulso, ilustrada na figura 6, possibilita controlar a velocidade com a qual o jabuti se move, pois um maior valor médio da tensão aplicada aos pinos que controlam a traduzir-se-á numa força maior aplicada nas rodas e, por conseguinte, numa maior velocidade.

Os pinos 19 e 12 da placa *Raspberry Pi* 3 B+, que possuem canais PWM independentes, foram conectados às rodas esquerda e direita do jabuti físico, respectivamente, e controlam ondas de tensão PWM de 10 KHz. O *duty cycle* de uma onda reflete configuração salva em arquivo JavaScript Object Notation (JSON), no mesmo diretório do

executável do *firmware*.

3.4.0.2 API

O modo como o *firmware* permite que o jabuti físico seja controlado externamente é através de uma interface de programação estabelecida através da rede local. O ambiente integrado, bem como a biblioteca *Python jabutixlib*, troca mensagens com o *firmware* via *WebSockets*, sendo que o cabeçalho dessas mensagens, constituído de um byte, indica o tipo da mensagem. Tais bytes de cabeçalho são sequenciais e foram convenientemente escolhidos para que o menor deles corresponda, em American Standard Code for Information Interchange (ASCII), à letra "A"minúscula.

Quadro 1 – Tipos de mensagens trocadas pela rede.

Byte identificador	**Mensagem**	**Enviada por**
0x61	Oferta de conexão *WebRTC*	Ambiente integrado
0x62	Resposta a oferta de conexão *WebRTC*	Jabuti físico
0x63	Comando de mover o jabuti à frente	Ambiente integrado
0x64	Comando de mover o jabuti à esquerda	Ambiente integrado
0x65	Comando de mover o jabuti à direita	Ambiente integrado
0x66	Comando de requisição à configuração dos movimentos	Ambiente integrado
0x67	Comando para alterar a configuração dos movimentos	Ambiente integrado
0x68	Informação de que o jabuti completou movimento à frente	Jabuti físico
0x69	Informação de que o jabuti completou movimento à esquerda	Jabuti físico
0x70	Informação de que o jabuti completou movimento à direita	Jabuti físico

Fonte: Elaborado pelo autor (2022).

No quadro 1 vê-se os diferentes tipos de mensagens trocadas. Certas mensagens, como as de comandos de mover o jabuti, não requerem informação adicional alguma e são, portanto, consistidas apenas do byte do cabeçalho. As mensagens relativas ao estabelecimento da conexão *WebRTC* requerem informação adicional (da *string* JSON representando a oferta ou a resposta *WebRTC*), e as mensagens relativas às configurações dos movimentos do jabuti também requerem informação adicional (*string* representando a configuração).

O acesso ao jabuti físico pode ser visto como chamadas remotas de função (Remote Procedure Call (RPC)). Um ponto importante no design é a sincronia nas funções relativas aos movimentos do jabuti: o ambiente integrado, durante a execução de um programa, envia a mensagem relativa a um movimento aguarda a confirmação, por parte do jabuti físico, de que o movimento foi concluído.

3.4.0.3 Particularidades

O modo como os dispositivos usuários identificam o endereço IP do jabuti é através do protocolo mDNS, um protocolo que objetiva resolver nomes de domínio em redes pequenas, sendo ele provido *out-of-the-box* pelo Raspbian.

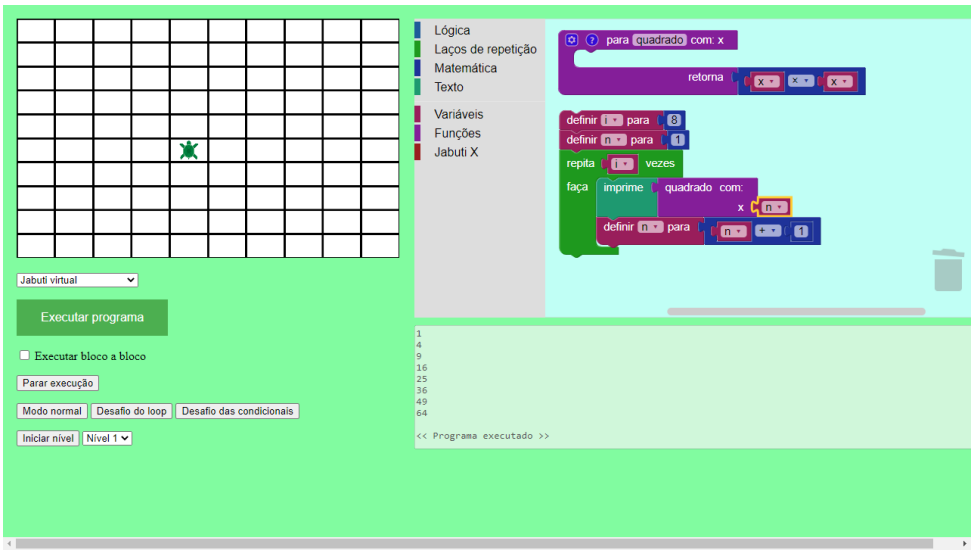
Diferentemente do projeto Jabuti Edu, em que o jabuti físico atua como um ponto de acesso, no trabalho desenvolvido o *firmware* conecta-se em uma rede comum à dos dispositivos usuários. Essa decisão de design faz com que o dispositivo conectado não fique sem conectividade à *Internet*. Em contrapartida, caso o jabuti seja movido de local, será necessário conectá-lo à rede do novo local.

3.4.1 Ambiente integrado

O ambiente integrado desenvolvido consiste do ambiente de programação em blocos, o qual o usuário efetivamente usa para programar, da saída padrão para o código do usuário e do mapa de blocos ou do visualizador (câmera).

Através do ambiente de programação, o usuário é capaz de estruturar um fluxo de instruções representadas por blocos. Ao executar seu programa, ele pode visualizar seu funcionamento através ou do jabuti físico ou do jabuti virtual.

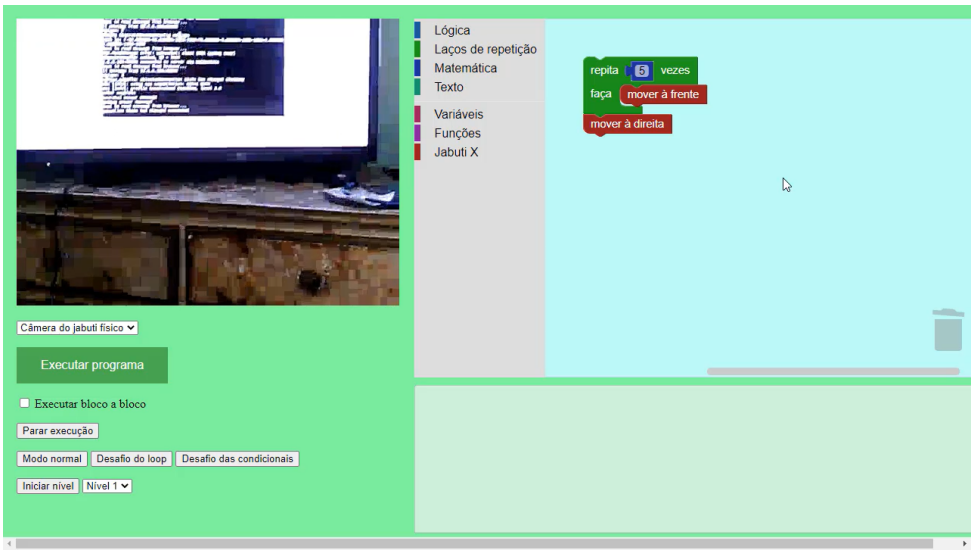
Figura 7 – Execução de simples programa.



Fonte: Elaborado pelo autor (2022).

A figura 7 exemplifica a tela vista pelo usuário ao término da execução de um simples programa. À esquerda, tem-se a área do mapa de blocos, em que o jabuti virtual está; à direita, vê-se o ambiente de programação em blocos e, abaixo, a saída-padrão para o programa (o programa em questão exibe, na saída-padrão, valores para o quadrado de alguns números).

Figura 8 – Usuário programando o jabuti físico.



Fonte: Elaborado pelo autor (2022).

Na circunstância em que o jabuti virtual está sendo utilizado, o ambiente de programação, ao invés de direcionar as instruções ao jabuti físico, as redireciona para o jabuti virtual. Se, por outro lado, o jabuti físico estiver sendo utilizado, o mapa de blocos é substituído por uma visão da câmera frontal do dispositivo (conforme demonstra figura 8).

Inicialmente cogitou-se desenvolver o ambiente como uma aplicação *desktop*, porém julgou-se ser mais adequado desenvolvê-lo como uma página *web*, primariamente pela acessibilidade.

Como trata-se de uma aplicação *web*, utilizou-se a linguagem *JavaScript*, que, além de *WebAssembly*, é a única linguagem capaz de ser executada nativamente por navegadores. HyperText Markup Language (HTML) foi usado para definir os elementos da página, e Cascading Style Sheets (CSS) para definir seus estilos.

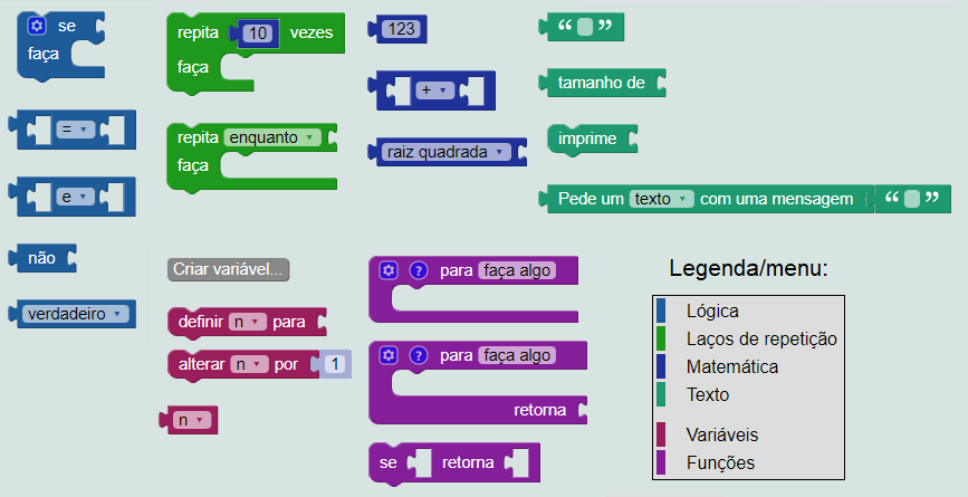
3.4.1.1 Linguagem em blocos

A fim de cumprir com os requisitos funcionais levantados, utilizou-se a biblioteca *frontend JavaScript Blockly*. Tal biblioteca permitiu colocar no ambiente um elemento *workspace*, que, na perspectiva do usuário, seja um local em que se estrutura um programa em blocos (pode ser visto nas figuras 7 e 8), e, na perspectiva do desenvolvedor, seja um elemento em que o usuário entre com código *JavaScript* sintaticamente correto (internamente a biblioteca traduz o programa que o usuário criou em código *JavaScript*). Para executar o código *JavaScript* que representa o programa criado pelo usuário, usou-se outra biblioteca, a JS-Interpreter, para interpretar o código.

Os blocos utilizados foram os padrões fornecidos como exemplo pela biblioteca

Blockly, que incluem blocos na categoria de lógica, de laço de repetição, de matemática, de texto, de variáveis e de funções.

Figura 9 – Blocos fornecidos como exemplo pela própria biblioteca Blockly.



Fonte: Elaborado pelo autor (2022).

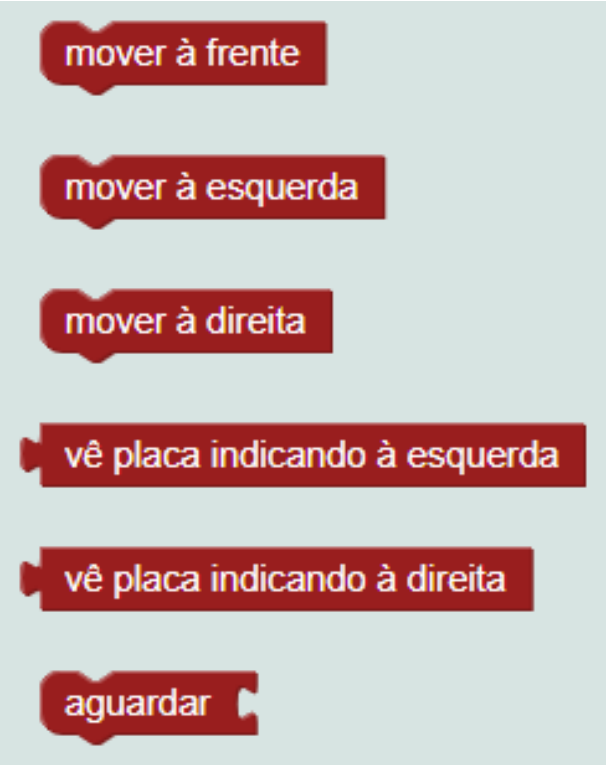
Os blocos base, que podem ser vistos na figura 9, constituem a linguagem utilizada pelo ambiente integrado. Tais blocos proveem a base necessária para qualquer linguagem de programação, como instruções condicionais (na aba de blocos lógicos) e funcionalidades para entrada e saída de dados.

Blockly é uma biblioteca suficientemente robusta a ponto do design de seus blocos poder ser projetado para evitar erros sintáticos por parte de seus usuários. De maneira naturalmente intuitiva, tal como um quebra-cabeça, blocos que retornam um valor, como os de entrada de dados, possuem, em seus lados esquerdos, uma parte extra, que serve como encaixe para blocos que recebem valores. Esses, por suas vezes, possuem um encaixe em seus lados direitos.

Certos blocos não possuem significado por si só, necessitando serem usados em conjunto com outros, como é o caso de uma operação aritmética (retorna um valor). Esses também convenientemente não possuem encaixes em suas partes superiores e inferiores, de modo que tentar encaixá-los inadequadamente seja impossível.

Além desses blocos, a fim de integrar funcionalidades referentes ao jabuti (tanto o físico quanto o virtual) ao ambiente, desenvolveu-se outros seis blocos sob a categoria "Jabuti X", vistos na figura 10.

Figura 10 – Blocos desenvolvidos.



Fonte: Elaborado pelo autor (2022).

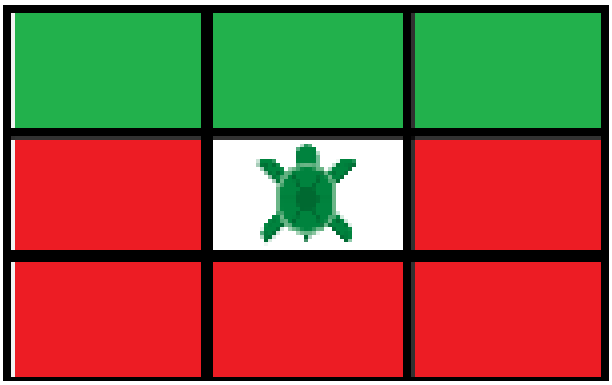
Os blocos de mover-se atuam no jabuti selecionado (o físico ou o virtual) e, como sugerem, o move. A execução do programa retoma apenas ao término do movimento. Os dois blocos subsequentes são blocos que retornam um valor *booleano* indicando se o piso à frente do jabuti virtual indica uma seta em uma determinada direção, sob a perspectiva do jabuti. O último bloco simplesmente faz com que o programa aguarde determinado tempo antes de prosseguir com a execução.

A linguagem disponibilizada pelo ambiente integrado consiste dos blocos padrão Blockly aliados aos desenvolvidos para os fins deste trabalho (descritos acima).

3.4.1.2 Jabuti virtual

Foi desenvolvida, através do paradigma da orientação a objetos, em um elemento HTML *canvas*, um mapa quadriculado de blocos e a representação de um jabuti virtual. O jabuti é capaz de se mover fluidamente, em movimentos discretos baseados nos blocos, ao longo do mapa. Da maneira como foi implementado, ao exceder a borda do mapa, o jabuti reaparece no canto oposto.

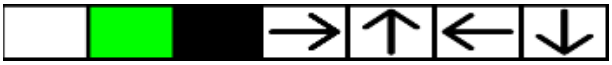
Figura 11 – Movimentos válidos para o jabuti virtual.



Fonte: Elaborado pelo autor (2022).

O jabuti virtual pode mover-se apenas para a frente ou para as diagonais, refletindo, assim, o comportamento simplificado do jabuti físico. A figura 11 demonstra os possíveis movimentos para um jabuti orientado para cima, em verde. Caso o movimento seja para a frente, a orientação do jabuti permanece sendo a mesma. Em caso de movimento à esquerda, sua orientação passa a ser para a esquerda e, analogamente, em caso de movimento à direita, sua orientação passa a ser para a direita.

Figura 12 – Diferentes pisos implementados no mapa.



Fonte: Elaborado pelo autor (2022).

O mapa é codificado para aparecer no elemento *canvas* de maneira dinâmica, com base no número de pisos, definido via código *JavaScript*. Ele ocupará sempre todo o *canvas* e, como consequência, mapas maiores fazem com que pisos apareçam menores na tela, assim como o jabuti virtual. Um piso no mapa pode ser um piso vazio, um piso de destino, um piso de parede, ou um piso de seta (como respectivamente ilustra figura 12).

3.4.1.3 Gamificações

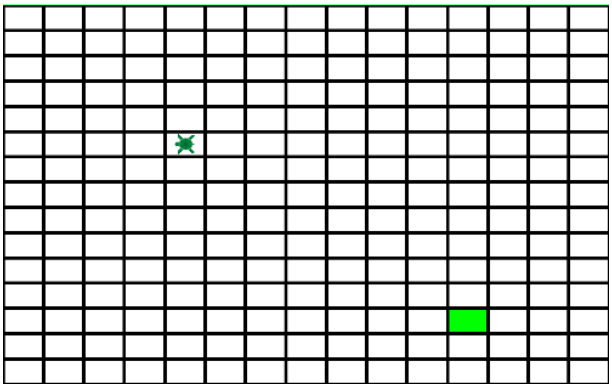
Para cumprir os requisitos funcionais de gamificação, foram desenvolvidos, com base no jabuti virtual, dois desafios: o "desafio do loop"e o "desafio das condicionais".

- ***Desafio do loop***

O desafio do loop propõe um jogo em que o objetivo é levar o jabuti até um determinado destino (denotado pelo piso em verde, conforme visto na figura 13). O

usuário não poderá, porém, usar mais que sete blocos em seu programa. O desafio é gerado proceduralmente de tal forma que garanta que seja impossível de ser resolvido sem o uso de estruturas de repetição, forçando o usuário a usar esse conceito de lógica de programação.

Figura 13 – Exemplo de mapa do desafio do loop, gerado proceduralmente.

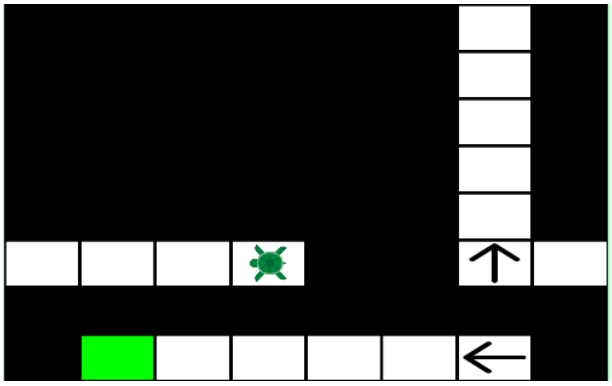


Fonte: Elaborado pelo autor (2022).

- ***Desafio das condicionais***

O desafio das condicionais, por sua vez, propõe um jogo com o mesmo objetivo. No entanto, o jabuti deve seguir um caminho específico (caso ele esbarre em uma parede, perde-se o jogo). O objetivo é elaborar um mesmo programa que sirva como solução a dois mapas distintos (similares ao da figura 14), gerados proceduralmente de forma que seja impossível que um mesmo programa seja solução a ambos sem que ele envolva o uso de instruções condicionais. Isso garante que o usuário apenas o consiga solucionar se fizer uso desse conceito de lógica de programação.

Figura 14 – Exemplo de mapa do desafio das condicionais, gerado proceduralmente.

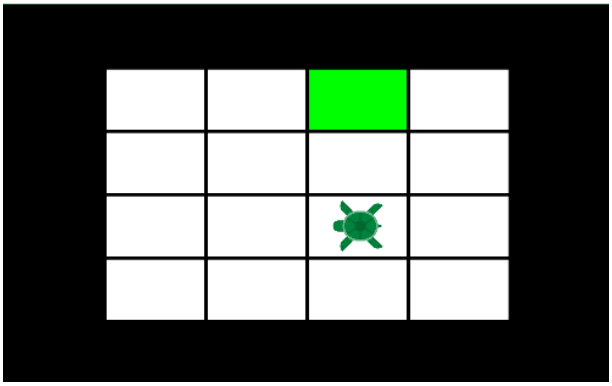


Fonte: Elaborado pelo autor (2022).

- ***Outros desafios***

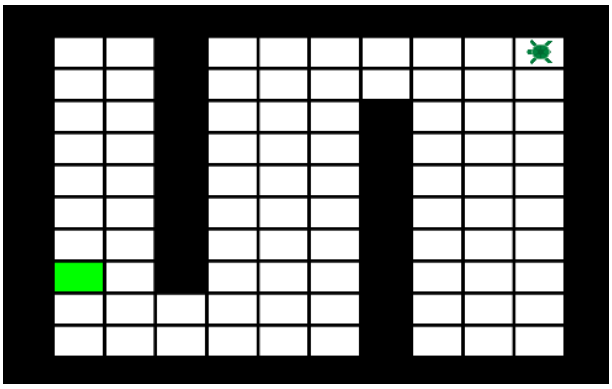
Além dos desafios, há gamificações mais simples, separadas em seis níveis distintos (ver figuras 15, 16 e 17). Em todos eles, com exceção do último, a borda do mapa é coberta por paredes. A partir do nível três, obstáculos de paredes são dispostos ao longo do mapa, de maneira a aumentar a dificuldade. O nível 6 (17) consiste de diversos obstáculos (paredes) dispostos ao longo do mapa, com probabilidade de 30% de um piso ser uma parede, porém de maneira que seja sempre possível solucioná-lo facilmente se levado em consideração que o jabuti, ao exceder a borda do mapa, reaparece no canto oposto.

Figura 15 – Exemplo de mapa do nível 1, gerado proceduralmente.



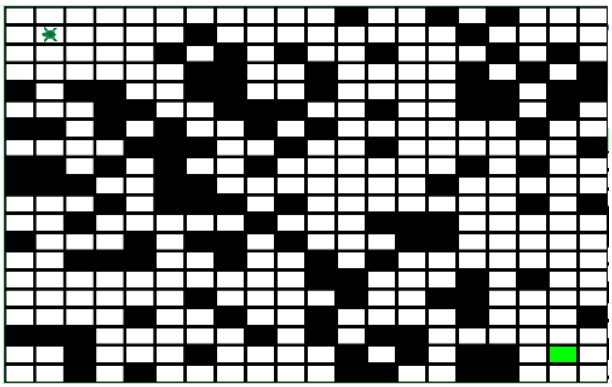
Fonte: Elaborado pelo autor (2022).

Figura 16 – Exemplo de mapa do nível 4, gerado proceduralmente.



Fonte: Elaborado pelo autor (2022).

Figura 17 – Exemplo de mapa do nível 6, gerado proceduralmente.



Fonte: Elaborado pelo autor (2022).

3.4.2 Biblioteca *Python*

Adicionalmente, foi desenvolvida uma biblioteca *Python* provendo uma classe capaz de interagir com o jabuti físico. Ela pode ser instalada através do comando "*pip install jabutixlib*".

A biblioteca pode ser útil no contexto em que um estudante tenha dominado a programação em blocos, podendo o professor instruí-lo a programar o jabuti físico através de código *Python*.

A figura 18 exemplifica seu uso em um programa que instancia a classe citada e faz uso dos métodos de movimentar o jabuti físico. Mais precisamente, o programa de exemplo move o jabuti físico três vezes para a frente, depois o move uma vez para a esquerda e, depois, uma vez para a direita.

Figura 18 – Código *Python* usando *jabutixlib*.

```
from jabutixlib import *

jabutixAPI = JabutiXAPI()

for i in range(0, 3):
    jabutixAPI.MoveForward()

jabutixAPI.MoveLeft()
jabutixAPI.MoveRight()
```

Fonte: Elaborado pelo autor (2022).

4 RESULTADOS E DISCUSSÕES

Através do trabalho relatado neste documento, foi possível portar o robô do projeto Jabuti Edu para um ambiente virtual *web*. Como o ambiente integrado consiste de arquivos estáticos, ele é facilmente difundido por meio da utilização de, por exemplo, um serviço CDN.

Com o uso do serviço citado, os arquivos estáticos que compõem o ambiente integrado podem ser fornecidos ao se acessar um domínio *web*, de modo que, no momento de escrita deste documento, é possível acessar o ambiente integrado através do *website* *https://jabutix.netlify.app/*. Tal acessibilidade faz-se benéfica em situações em que o acesso a instrumentos ou locais de ensino estejam limitados, como foi o caso com a pandemia de COVID-19.

A utilização de linguagem em blocos elimina por completo erros sintáticos e elimina grande parte dos erros léxicos (com exceção de, por exemplo, nomes de variáveis digitadas incorretamente ou erros de digitação de *strings*. Desta maneira, são enfatizados erros semânticos, erros esses que são, de fato, problemáticos durante o aprendizado de programação.

Em um teste realizado, um usuário relatou que uma dificuldade encontrada foi o nível 6 de gamificação, que foi propositalmente concebido para ser difícil caso não se perceba o comportamento do jabuti virtual ao exceder as bordas do mapa. Aspectos como esse e sensações de sucesso ao completar os desafios podem ser interpretados como algo frustrante ou como algo satisfatório, como um "clique"de percepção; não foi feito um estudo mais aprofundado com relação a isso em virtude do distanciamento social que se deu devido à pandemia de COVID-19. Entretanto, independentemente do modo que o usuário interprete o completar de um desafio, é garantido, por *design*, que ele usou conceitos universais de programação.

Os resultados dos esforços apresentados neste documento estão disponíveis, sob licença MIT, em *github.com/DaviFN/jabutix-env*, *github.com/DaviFN/jabutix-firmware* e *github.com/DaviFN/jabutixlib*.

5 CONCLUSÃO E TRABALHOS FUTUROS

Habilidade essencial à sociedade moderna, a programação vai muito além do código. Há quem subestime linguagens de programação em blocos, mas há de se lembrar que as linguagens de programação são ultimamente ferramentas para a expressão do raciocínio e a criatividade humana. Assim como, a exemplo, a linguagem C pode ser mais adequada para o desenvolvimento de aplicações embarcadas e a linguagem R, para aplicações estatísticas, linguagens de programação em blocos têm o importante papel de servirem como ferramentas de aprendizado.

O ambiente desenvolvido permite a codificação de programas em linguagem de blocos e a instantânea visualização de seu resultado no jabuti virtual ou no físico, e provê desafios cujas soluções requerem pensamento abstrato e uso de conceitos básicos de programação. Portanto, conclui-se que os objetivos específicos designados para o desenvolvimento do trabalho foram cumpridos.

Os robôs mencionados, tanto a adaptação feita por estudantes da Universidade Federal de Santa Catarina (UFSC) - Campus Araranguá quanto o robô do projeto Jabuti Edu, da UFRGS, sem adaptações, possuem custo inferior a alternativas comerciais de robótica educacional, sendo aproximadamente dez vezes mais barato que o conjunto Lego Mindstorms Ev3. Isso, aliado à ubiquidade de redes IEEE 802.11 (WiFi) e a ser possível utilizar o ambiente mesmo na ausência do robô físico, são pontos bastante relevantes no que diz respeito à acessibilidade do ambiente proposto.

O protótipo desenvolvido apresenta, no estado documentado, muitas limitações, passíveis de serem abordadas em futuros trabalhos. A exemplo, é apenas possível conectar-se a um jabuti físico em uma mesma rede local, não sendo tratados casos com múltiplos jabutis físicos. Ademais, como trata-se de um protótipo, há diversos *bugs*, situação recorrente no desenvolvimento de *software*, que evidenciar-se-iam a partir do momento que mais pessoas começassem a utilizar o sistema.

Embora os objetivos propostos tenham sido alcançados, a usabilidade do protótipo é comprometida pois inexiste uma página possibilitando ao instrutor um controle do uso do dispositivo, uma melhoria que poderia ser feita no trabalho. Uma melhoria no ambiente integrado a partir do próprio servidor estabelecido pelo jabuti físico como forma adicional de acessar o ambiente integrado, abordagem que não ocasionaria perda alguma de performance pois, uma vez que os arquivos estáticos do ambiente integrado estiverem carregados no navegador do dispositivo do usuário, toda a execução do ambiente dar-se-ia a partir de lá. Outra possível melhoria seria a integração de sensores ao jabuti possibilitando-o detectar o equivalente a pisos de parede e pisos de setas, expandindo a gamificação para o ambiente físico, bem como a implementação de blocos com funcionalidades de visão computacional tendo como base o *stream* de vídeo da câmera do dispositivo para, a exemplo, reconhecimento de textos como entrada de dados.

Mesmo no protótipo, porém, mostrou-se, ao longo do desenvolvimento, ser possível gradualmente apresentar conceitos de programação de uma maneira lúdica, beirando ao entretenimento. O estado documentado do ambiente integrado poder ser acessado virtualmente de qualquer lugar, bastando uma conexão à *Internet*, faz com que, mesmo em forma de protótipo, os esforços envolvidos neste trabalho possam ser utilizados para o aprendizado de programação de forma extremamente acessível.

REFERÊNCIAS

ALIMISIS, D. Educational robotics: Open questions and new challenges. *Themes in Science and Technology Education*, v. 6, n. 1, p. 63–71, 2013. Citado na página [19].

ALVES E.; ALVES, S. V. L. C. L. J. R. Módulo de programação baseada em blocos para a plataforma jabuti edu com blockly. *XXIII Congresso Internacional de Informática na Educação (TISE). Anais Nuevas Ideas em Informática Educativa*, v. 13, p. 641–647, 2017. Citado na página [23].

BACHILLER-BURGOS, P. et al. Learnblock: A robot-agnostic educational programming tool. *IEEE Access*, IEEE, v. 8, p. 30012–30026, 2020. Citado na página [20].

BILOTTA, E. et al. Edutainment robotics as learning tool. In: *Transactions on edutainment III*. [S.l.]: Springer, 2009. p. 25–35. Citado na página [19].

CASINADER, N. *One giant step for. . . ? The moon landing and its legacy for geography?* [S.l.]: Taylor & Francis, 2019. 175–177 p. Citado na página [11].

CROSS, J. et al. A visual robot-programming environment for multidisciplinary education. In: IEEE. *2013 IEEE International Conference on Robotics and Automation*. [S.l.], 2013. p. 445–452. Citado na página [19].

FAGIN, B.; MERKLE, L. Measuring the effectiveness of robots in teaching computer science. *Acm sigcse bulletin*, ACM New York, NY, USA, v. 35, n. 1, p. 307–311, 2003. Citado na página [19].

FEDERICI, S. A minimal, extensible, drag-and-drop implementation of the c programming language. In: *Proceedings of the 2011 conference on Information technology education*. [S.l.: s.n.], 2011. p. 191–196. Citado (2) vezes nas páginas [12 e 20].

FEURZEIG, W.; LUKAS, G. Logo—a programming language for teaching mathematics. *Educational Technology*, JSTOR, v. 12, n. 3, p. 39–46, 1972. Citado na página [14].

FRASER, N. Ten things we've learned from blockly. In: IEEE. *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*. [S.l.], 2015. p. 49–50. Citado na página [18].

FU, J. et al. Karenao: A tangible block-based programming environment. In: IEEE. *2021 18th International Conference on Ubiquitous Robots (UR)*. [S.l.], 2021. p. 314–319. Citado na página [21].

GIANG, V. *'Gamification' Techniques Increase Your Employees' Ability To Learn By 40%*. 2013. Disponível em: <http://whttp://www.businessinsider.com/gamification-techniques-increase-your-employeesability-to-learn-by-40-2013-9>. Citado na página [17].

GOOGLE. *Try Blockly*. 2021. Disponível em: <https://developers.google.com/blockly>. Citado (2) vezes nas páginas [15 e 17].

ISLAM, A. et al. Edubot: An educational robot for underprivileged children. In: IEEE. *2019 International Conference on Automation, Computational and Technology Management (ICACTM)*. [S.l.], 2019. p. 232–236. Citado na página [22].

JONES, B. A. et al. Gamification of dietary decision-making in an elementary-school cafeteria. *PloS one*, Public Library of Science San Francisco, USA, v. 9, n. 4, p. e93872, 2014. Citado na página [17].

KAPP, K. M. *The gamification of learning and instruction: game-based methods and strategies for training and education*. [S.l.]: John Wiley & Sons, 2012. Citado na página [17].

KELLEHER, C.; PAUSCH, R.; KIESLER, S. Storytelling alice motivates middle school girls to learn computer programming. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. [S.l.: s.n.], 2007. p. 1455–1464. Citado na página [18].

KIRYAKOVA, G.; ANGELOVA, N.; YORDANOVA, L. Gamification in education. In: PROCEEDINGS OF 9TH INTERNATIONAL BALKAN EDUCATION AND SCIENCE CONFERENCE. [S.l.], 2014. Citado na página [18].

KURLAND, D. M. et al. A study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research*, SAGE Publications Sage CA: Los Angeles, CA, v. 2, n. 4, p. 429–458, 1986. Citado na página [11].

LEWIS, C. M. How programming environment shapes perception, learning and goals: logo vs. scratch. In: *Proceedings of the 41st ACM technical symposium on Computer science education*. [S.l.: s.n.], 2010. p. 346–350. Citado na página [16].

MAIA, R. de O. et al. Desenvolvimento de um dispositivo para apoio ao ensino de computação e robótica. In: *Colloquium Exactarum. ISSN: 2178-8332*. [S.l.: s.n.], 2014. v. 6, n. 2, p. 71–85. Citado na página [22].

MALONEY, J. et al. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, ACM New York, NY, USA, v. 10, n. 4, p. 1–15, 2010. Citado na página [16].

MERKOURIS, A.; CHORIANOPOULOS, K.; KAMEAS, A. Teaching programming in secondary education through embodied computing platforms: Robotics and wearables. *ACM Transactions on Computing Education (TOCE)*, ACM New York, NY, USA, v. 17, n. 2, p. 1–22, 2017. Citado na página [12].

PAPERT, S. Mindstorms: Children, computers, and powerful ideas. *Harvester Press (Unitend Kingdom). DOI*, v. 10, p. 978–3, 1980. Citado na página [14].

PASTERNAK, E.; FENICHEL, R.; MARSHALL, A. N. Tips for creating a block language with blockly. In: IEEE. *2017 IEEE Blocks and Beyond Workshop (B&B)*. [S.l.], 2017. p. 21–24. Citado na página [17].

PEA, R. D. Logo programming and problem solving. 1987. Citado na página [14].

PITEIRA, M.; HADDAD, S. R. Innovate in your program computer class: an approach based on a serious game. In: *Proceedings of the 2011 Workshop on Open Source and Design of Communication*. [S.l.: s.n.], 2011. p. 49–54. Citado na página [11].

PRICE, T. W.; BARNES, T. Comparing textual and block interfaces in a novice programming environment. In: *Proceedings of the eleventh annual international conference on international computing education research*. [S.l.: s.n.], 2015. p. 91–99. Citado na página [16].

RESNICK, M. et al. Scratch: Programming for all. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 52, n. 11, p. 60–67, nov 2009. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/1592761.1592779>. Citado na página [15].

SILVA, P. F. da; ROCKENBACK, E. J.; FAGUNDES, L. da C. Jabuti edu: uma plataforma de robótica educacional de baixo custo. 2021. Citado na página [23].

SOUZA, D. M.; BATISTA, M. H. da S.; BARBOSA, E. F. Problemas e dificuldades no ensino de programação: Um mapeamento sistemático. *Revista Brasileira de Informática na Educação*, v. 24, n. 1, p. 39, 2016. Citado na página [11].

TEMPEL, M. Blocks programming. *CsTA Voice*, v. 9, n. 1, p. 3–4, 2013. Citado na página [15].

TVONTARIO. *Bits and Bytes Episode 6: Computer Languages*. 1983. Disponível em: <https://archive.org/details/bits_and_bytes_6>. Citado na página [14].

UFRGS, I. de I. *ROBO+EDU*. 2015. Disponível em: <https://www.ufrgs.br/robomaisedu/>. Citado na página [23].

WEINGAERTNER, T. et al. Smart contracts using blockly: Representing a purchase agreement using a graphical programming language. In: IEEE. *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. [S.l.], 2018. p. 55–64. Citado na página [15].